IN THE CLAIMS

Please amend the claims as indicated in the rewritten claims listed below:

Claim Amendments:

Claim 1 (Canceled)

Claim 2 (Currently Amended): A method for protecting a computer from unauthorized code, the computer including at least one processor that executes instructions stored in a memory, the memory being organized into separately addressable memory blocks, the method comprising:

executing a program, the program having a series of computer-executable instructions;

verifying that the program is valid, the program being valid when the program does not include the unauthorized code;

ensuring that the program is not executed without dynamically performing the verifying, the verifying based on a next instruction to be executed in the series of computer-executable instructions in the program; and

continuing execution of the program after dynamically performing the verifying by executing the next instruction as long as the verifying determines that the program is valid and generating a protective response when the verifying determines that the program is not valid;

wherein the verifying that the program is valid comprises:

identifying a the next instruction ~~of the series of computer-executable instructions~~ to be executed ~~when executing~~ in the program;

for the next instruction to be executed in ~~, and during the executing of~~ the program, determining an identifying value for a memory block that contains the next instruction;

determining~~, during the executing of the program,~~ whether the identifying value satisfies a validation condition, wherein the determining as to whether the identifying value satisfies the validation condition requires comparing the identifying value of the memory block with a set of reference values; and

determining that the program is valid when the validation condition is satisfied.

Claim 3 (Previously Presented): The method of claim 2, wherein the validation condition is that the identifying value of the memory block matches any reference value in the set of reference values.

Claim 4 (Previously Presented): The method of claim 2, wherein the validation condition is that the identifying value of the memory block differs from each reference value in the set of reference values.

Claim 5 (Canceled)

Claim 6 (Currently Amended): A method for protecting a computer from unauthorized code, the computer including at least one processor that executes instructions stored in memory, the memory being organized into separately addressable memory blocks, the method comprising:

executing a program, the program having a series of computer-executable instructions;

verifying that the program is valid as the program executes, the program being valid when the program does not include the unauthorized code, the verifying based on a current instruction to be executed when executing the series of instructions in the program; ~~and~~

generating a protective response when the verifying determines that the program is not valid; and

executing the current instruction when the verifying determines that the program is valid;

wherein the verifying that the program is valid comprises:

identifying ~~a~~ the current instruction to be executed when executing the series of instructions, the current instruction being one of the series of instructions being executed identified for submission to the processor for execution and not yet executed at a time of the identifying;

for at least the current instruction, computing a hash value as a function of a subset of contents of a current memory block that contains the current instruction;

determining, during the executing of the series of instructions, whether the hash value satisfies a validation condition by comparing the hash value of the current memory block with a set of reference values; and

determining that the program is valid when the hash value satisfies the validation condition, and determining that the program is not valid when the hash value does not satisfy the validation condition;

wherein the computing of the hash value comprises applying a mask to the current memory block, the mask being a data structure that designates at least one byte of the current memory block to be ignored in the computing of the hash value, the data structure designating less than an entire memory block so that the hash value is based on only part of the contents of the current memory block.

Claim 7 (Previously Presented): The method of claim 6, further comprising:

identifying, an indeterminate portion of the current memory block, the indeterminate portion being non-indicative of validity of the current memory block as a whole; and

configuring the mask so that the mask designates at least the indeterminate portion to be ignored when generating the hash value.

Claim 8 (Previously Presented): The method of claim 2, wherein the verifying that the program is valid further comprises:

for each of the separately addressable memory blocks, indicating in a structure whether the memory block is valid;

accessing the structure to determine whether the memory block is valid prior to the determining of the identifying value; and

performing the determining of the identifying value when the structure does not indicate that the memory block is valid and directly allowing execution of the next instruction when the structure indicates that the memory block is valid.

Claim 9 (Previously Presented): The method of claim 8, wherein the structure comprises a group of hardware attribute indicators, and wherein the indicating in the structure whether the plurality of memory blocks is validated comprises setting one of the hardware attribute indicators, the one hardware attribute indicator corresponding to the memory block.

Claim 10 (Currently Amended): ~~A~~ The method ~~as in~~ of claim 9, in which the hardware attribute indicators are execute and write permission attributes associated with an entry in a translation lookaside buffer.

Claim 11 (Previously Presented): The method of claim 8, wherein the structure comprises a software data structure, and wherein the indicating in the structure whether the plurality of memory blocks is validated comprises making a corresponding entry in the software data structure.

Claim 12 (Previously Presented): The method of claim 8, wherein the determining of the identifying value for the memory block and the determining of whether the identifying value satisfies the validation condition are performed only when the structure does not indicate that the memory block is valid, the method further comprising:

when the structure does not indicate that the memory block is valid, modifying the structure to indicate that the memory block is valid when the identifying value satisfies the validation condition.

Claim 13 (Previously Presented): The method of claim 12, further comprising:

sensing modification of one of the memory blocks that the structure indicates is valid and, in response to the modification, setting its indication in the structure to indicate that the memory block is not valid.

Claim 14 (Previously Presented): The method of claim 8, wherein the verifying that the program is valid further comprises:

determining a branch history for the next instruction; and

checking whether the memory blocks in which instructions in the branch history are located are valid, the validation condition including the requirement that each checked memory block in the branch history is valid.

Claim 15 (Previously Presented): The method of claim 2, wherein the determining of the identifying value and the determining as to whether the validation condition has been satisfied are performed only after a triggering event occurs.

Claim 16 (Previously Presented): The method of claim 15, wherein the triggering event is writing of at least one new unit of code or data to any physical component within the computer.

Claim 17 (Previously Presented): The method of claim 15, in which the triggering event is an attempted execution of any instruction located on any unverified memory block.

Claim 18 (Previously Presented): The method of claim 15, wherein the triggering event is an attempted execution of any instruction located on any unverified memory block of newly installed software.

Claim 19 (Previously Presented): The method of claim 15, further comprising triggering the verification of the computer instructions depending on an identity of a user of the computer, the user having caused the next instruction to be identified for execution.

Claim 20 (Canceled)

Claim 21 (Previously Presented): The method of claim 15, further comprising triggering dynamic verification depending on a context in which the next instruction is submitted for execution, wherein the context is a level of security clearance associated with the computer, a user of the computer, or a program of which the next instruction is a part.

Claim 22 (Previously Presented): The method of claim 2, wherein the identifying of the next instruction is performed for only a sample of the series of instructions.

Claim 23 (Previously Presented): The method of claim 22, wherein the sample is a time-sampled subset of the series of instructions.

Claim 24 (Previously Presented): The method of claim 22, wherein the sample is a sequentially sampled subset of the series of instructions.

Claim 25 (Previously Presented): The method of claim 22, wherein the sample is a subset of the series of instructions sampled spatially, the sampling being over a range of memory block identifiers.

Claim 26 (Previously Presented): The method of claim 2, wherein the protective response comprises termination of a software entity with which the current memory block is associated.

Claim 27 (Previously Presented): The method of claim 2, wherein the protective response comprises suspension of execution of a software entity with which the current memory block is associated.

Claim 28 (Previously Presented): The method of claim 2, wherein the protective response comprises a message posted to a user, system administrator, or other predetermined recipient.

Claim 29 (Previously Presented): The method of claim 2, wherein:

the computer includes a virtual machine running in a direct execution mode on an underlying hardware platform via an intermediate software layer; and

the protective response comprises a switching of an execution mode of the virtual machine from the direct execution mode to a binary translation mode.

Claim 30 (Previously Presented): The method of claim 2, wherein:

the computer includes a virtual machine running on an underlying hardware platform via an intermediate software layer; and

the protective response includes checkpointing the state of the virtual machine.

Claim 31 (Currently Amended): The method of claim 2, wherein the protective response is a first possible response, the method further comprising:

associating the first possible response with the memory block;

associating a second possible response with a different memory block; and

upon detection of failure of the next instruction to satisfy the validation condition, identifying which one of the possible responses is associated with the memory block, and generating the one possible response associated with the memory block in which the next instruction is located.

Claim 32 (Previously Presented): The method of claim 2, further comprising:

associating reference values from the set of reference values with respective programs such that each association signifies that the reference value corresponds to a memory block storing instructions for one of the programs; and

tracking which of the respective programs is being executed and the association between the matching reference value and the corresponding one of the programs.

Claim 33 (Previously Presented): The method of claim 2, wherein:

the computer includes a virtual machine (VM) running on an underlying hardware platform via an intermediate software layer operable to switch the virtual machine between a direct execution mode and a binary translation mode; and

the series of instructions comprise VM-issued instructions issued in conjunction with binary translation of any of the VM-issued instructions, the VM-issued instructions thereby being verified.

Claim 34 (Canceled)

Claim 35 (Currently Amended): A machine readable storage medium embodying executable code for protecting a computer from unauthorized code, the computer including at least one processor that executes instructions stored in a memory, the memory being organized into separately addressable memory blocks, the executable code being a verification engine causing the computer to perform a method having operations of:

executing a program, the program having a series of computer-executable instructions;

verifying that the program is valid, the program being valid when the program does not include the unauthorized code;

ensuring that the program is not executed without dynamically performing the verifying, the verifying based on a next instruction to be executed in the series of computer-executable instructions in the program; and

continuing execution of the program after dynamically performing the verifying by executing the next instruction as long as the verifying determines that the program is valid and generating a protective response when the verifying determines that the program is not valid;

wherein the verifying that the program is valid comprises:

identifying ~~a~~ the next instruction of the series of computer-executable instructions to be executed when executing the program;

for the next instruction, and during the execution of the program, determining an identifying value for a memory block that contains the next instruction;

determining whether the identifying value satisfies a validation condition, wherein the determining as to whether the identifying value satisfies the validation condition requires comparing the identifying value of the memory block with a set of reference values; and

_____determining that the program is valid when the validation condition is satisfied.

Claim 36 (Previously Presented): The machine readable storage medium of claim 35, wherein the validation condition is that the identifying value of the memory block matches any reference value in the set of reference values.

Claim 37 (Previously Presented): The machine readable storage medium of claim 35, wherein the validation condition is that the identifying value of the memory block differs from each reference value in the set of reference values.

Claim 38 (Previously Presented): The machine readable storage medium of claim 35, wherein: the identifying value is a hash value that is computed as a function of contents of the memory block.

Claim 39 (Previously Presented): The machine readable storage medium of claim 38, wherein some of the contents of the memory block are ignored when computing the hash value, the ignored portion being defined by a subset selection structure.

Claim 40 (Previously Presented): The machine readable storage medium of claim 39, wherein the subset selection structure is a mask.

Claim 41 (Previously Presented): The machine readable storage medium of claim 35, wherein the verifying that the program is valid further comprises:

for each of the separately addressable memory blocks, indicating in a structure whether the memory block is valid;

accessing the structure to determine whether the memory block is valid prior to the determining of the identifying value; and

performing the determining of the identifying value when the structure does not indicate that the memory block is valid and directly allowing execution of the next instruction when the structure indicates that the memory block is valid.

Claim 42 (Previously Presented): The machine readable storage medium of claim 41, wherein the structure comprises a group of hardware attribute indicators, and wherein the indicating in the structure whether the plurality of memory blocks is valid comprises setting one of the hardware attribute indicators corresponding to the memory block.

Claim 43 (Previously Presented): The machine readable storage medium of claim 42, in which the hardware attribute indicators are execute and write permission attributes associated with an entry in a translation lookaside buffer.

Claim 44 (Previously Presented): The machine readable storage medium of claim 41, wherein the structure comprises a software data structure, and wherein the verifying further comprises selecting for verification only a sample of the next instructions.

Claim 45 (Canceled)

Claim 46 (Previously Presented): The machine readable storage medium of claim 35, wherein:

the verification engine resides in an intermediate virtualization layer between a virtual machine and a hardware platform of the computer;

the next instruction is issued by the virtual machine; and

the verifying that the program is valid is performed while copying or translating in conjunction with binary translation of instructions for the virtual machine.

Claim 47 (Previously Presented): The machine readable storage medium of claim 46, wherein the protective response comprises switching the intermediate virtualization layer to a binary translation mode.

Claim 48 (Previously Presented): The method of claim 2, wherein the identifying value is a hash value that is computed as a function of all contents of the memory block.

Claim 49 (Previously Presented): The method of claim 2, wherein the identifying value is a hash value that is computed as a function of contents of the memory block such that some of the contents of the memory block are ignored when computing the hash value, the ignored portion being defined by a subset selection structure.

Claim 50 (Currently Amended): A method for verifying the validity of instructions in a computer that includes at least one physical processor that executes instructions stored in a memory of the computer, the memory being organized into separately addressable memory blocks, the computer also including system software and verification software, the system software performing hardware interface, resource-allocating, and control functions of an operating system, the method comprising:

monitoring instructions to be executed under control of the system software;

as long as the instructions to be executed under control of the system software are stored in one or more memory blocks for which validation is deemed unnecessary, allowing the instructions to be executed under control of the system software; and

detecting that an unvalidated instruction is to be executed under control of the system

software, the unvalidated instruction being stored in a memory block that has not been validated, and, before allowing the unvalidated instruction to execute, attempting to validate the unvalidated memory block by performing the following steps:

determining an identifying value for the unvalidated memory block;

comparing the identifying value of the unvalidated memory block with a set of reference values;

if the identifying value satisfies a validation condition, allowing execution after satisfying the validation condition, under control of the system software, of instructions stored in the unvalidated memory block; and

if the identifying value does not satisfy the validation condition, generating a response.

Claim 51 (Previously Presented): The method of claim 50, wherein the validation condition is that the identifying value of the memory block matches any reference value in the set of reference values.

Claim 52 (Previously Presented): The method of claim 50, wherein the validation condition is that the identifying value of the memory block differs from each reference value in the set of reference values.

Claim 53 (Previously Presented): The method of claim 50, wherein the determining of the identifying value comprises computing a hash value as a function of contents of the unvalidated memory block, the identifying value being the hash value.

Claim 54 (Previously Presented): The method of claim 53, wherein some of the contents of the unvalidated memory block are ignored when computing the hash value, the ignored portion being defined by a subset selection structure.

Claim 55 (Previously Presented): The method of claim 54, wherein the subset selection structure is a mask.

Claim 56 (Previously Presented): The method of claim 50, wherein the detecting that an unvalidated instruction is to be executed comprises:

for each of the separately addressable memory blocks, indicating in a structure whether the memory block is valid;

accessing the structure to determine whether one of the memory blocks containing the instructions to be executed is valid prior to the determining of the identifying value; and

detecting that the instruction to be executed is unvalidated when the structure does not indicate that the memory block that contains the instruction to be executed is valid.

Claim 57 (Previously Presented): The method of claim 56, wherein the structure comprises a group of hardware attribute indicators, and wherein the indicating in the structure whether the memory block is valid comprises setting one of the hardware attribute indicators corresponding to the memory block.

Claim 58 (Previously Presented): The method of claim 56, wherein the structure comprises a software data structure.

Claim 59 (Previously Presented): The method of claim 50, wherein:

the verification software resides in an intermediate virtualization layer between a virtual machine and a hardware platform of the computer;

the instructions to be executed that are being monitored comprise instructions issued by the virtual machine; and

the verifying of the validity of the instructions is carried out while copying or translating in conjunction with binary translation of the instructions to be executed.

Claim 60 (Previously Presented): The method of claim 50, wherein the detecting that an unvalidated instruction is to be executed further comprises:

determining a branch history for the unvalidated instruction; and

checking whether memory blocks in which instructions in the branch history are located are valid, the validation condition including the requirement that each checked memory block in the branch history is valid.

Claim 61 (Currently amended): ~~the~~ The method of claim 50, wherein monitoring of the instructions to be executed and the detecting that the ~~an~~ unvalidated instruction is to be executed are performed only after a triggering event occurs.

Claim 62 (New): The method of claim 2, wherein ensuring that the program is not executed without dynamically performing the verifying comprises interrupting execution of the series of computer-executable instructions of the program while dynamically performing the verifying.